Hierarchical RL and Transfer Learning
Used Materials

- **Disclaimer**: Some of the material was provided by Tejas D. Kulkarni and Emilio Parisotto
Talk Roadmap

• Hierarchical Deep RL

• Transfer Learning

• Learning with Memory
Hierarchical RL

- Flat RL works well but on small problems
- To Scale-up: Decompose large problems into smaller ones
- Transfer: Share/reuse tasks
Typical RL Setup

- Where do goals/rewards come from?

- What should an agent do in the off-time when there are no external rewards? **Unsupervised learning** for sensory motor knowledge?
  - **Curiosity, self-play** in animals and children
  - Solving for **temporally extended intrinsic rewards** in the space of sensor and feature values (visual, auditory, CNN features etc.) can provide a rich basis set of behaviors.
  - These behaviors can then be recombined or repurposed for **sparsely defined** real tasks.

- The basic abstraction needed to build towards this from an RL perspective is called an option (Sutton 1999)
Hierarchical Deep RL

Meta controller uses a DNN to learn an action-value epsilon-greedy policy over intrinsic goals.

Controller uses a DNN to learn an action-value epsilon-greedy policy over actions to satisfy an intrinsic goal.

- Deep RL agents with a temporally extended exploration policy can achieve good results whenever the agent has access to a compact goal space.
Types of Goal

- **Extrinsic reward** functions provided by the environment

- **Intrinsic motivation**, curiosity and self-play in the space of agent’s sensory experiences

- Goals can also be posed in the space of internal spatial and/or temporal representations:
  - `<object1, relation, object2>` or `<self, go-to, ladder>`

- Goals can be extracted using structural decomposition of the environment or learning dynamics (information-theoretic)
Example: Montezuma’s Revenge

Games MZ are good test beds to evaluate a taxonomy of intrinsically motivated RL agents.
MDPs and Semi-MDPs

Options + MDP = Semi MDP

Kulkarni et al., NIPS 2016, Sutton et al., 1999
Semi MDP

Meta-controller:

\[ Q^*_2(s, g) = \max_{\pi_g} \mathbb{E}\left[ \sum_{t'+t} f_{t'} + \gamma \max_{g'} Q^*_2(s_{t+N}, g') \mid s_t = s, g_t = g, \pi_g \right] \]

- \( N \): the number of time steps until the controller halts given the current goal, \( g \)

- \( \pi_g \) is the policy over goals.

- \( f_t \) are reward signals received from the environment.

The meta-controller looks at the raw states and produces a policy over goals by estimating the action-value function \( Q_2 \) (to maximize expected future extrinsic reward).
Semi MDP

- **Meta-controller:**

  \[ Q_2^*(s, g) = \max_{\pi_g} \mathbb{E} \left[ \sum_{t'=t}^{t+N} f_{t'} + \gamma \max_{g'} Q_2^*(s_{t+N}, g') \mid s_t = s, g_t = g, \pi_g \right] \]

- **Controller:**

  \[ Q_1^*(s, a; g) = \max_{\pi_{ag}} \mathbb{E} \left[ \sum_{t'=t}^{\infty} \gamma^{t'-t} r_{t'} \mid s_t = s, a_t = a, g_t = g, \pi_{ag} \right] \]

  \[ = \max_{\pi_{ag}} \mathbb{E} \left[ r_t + \gamma \max_{a_{t+1}} Q_1^*(s_{t+1}, a_{t+1}; g) \mid s_t = s, a_t = a, g_t = g, \pi_{ag} \right] \]

- The controller takes in states and the current goal, and produces a **policy over actions** by estimating the action-value function \( Q_1 \) to solve the predicted goal (by maximizing expected **future intrinsic reward**).
Semi MDP

- Meta-controller:

\[
Q_2^*(s, g) = \max_{\pi_g} E \left[ \sum_{t'=t}^{t+N} f_{t'} + \gamma \max_{g'} Q_2^*(s_{t+N}, g') \mid s_t = s, g_t = g, \pi_g \right]
\]

- Controller:

\[
Q_1^*(s, a; g) = \max_{\pi_{ag}} E \left[ \sum_{t'=t}^{\infty} \gamma^{t'-t} r_{t'} \mid s_t = s, a_t = a, g_t = g, \pi_{ag} \right]
\]

\[
= \max_{\pi_{ag}} E \left[ r_t + \gamma \max_{a_{t+1}} Q_1^*(s_{t+1}, a_{t+1}; g) \mid s_t = s, a_t = a, g_t = g, \pi_{ag} \right]
\]

- Solve for \( Q_1 \) and \( Q_2 \) using separate Deep Q-Networks, replay buffers and using TD-learning via SGD at different time scales. \( Q_1 \) ticks much faster than \( Q_2 \)
Example: Montezuma’s Revenge

DQN

h-DQN with options constructed from a set of pre-defined primitives
Example: Montezuma’s Revenge

goal visit statistic

extrinsic rewards
Talk Roadmap

- Hierarchical Deep RL
- Transfer Learning
- Learning with Memory
Motivation

- Mnih et al., 2014: Learn complex policies directly from raw pixel data using a Deep Q-Network (DQN).

- Despite using the same hyperparameters, a separate DQN was trained for each game.

- Can a single network be trained that can play many games at once, at a near-expert level?

- Why do we need multitask?
  - Transfer: Can potentially learn new games faster if the model can leverage knowledge about the previous games it learnt.
  - Test-time efficiency: we only need a single network.
Learning as a Function of Time

Can learn new games faster by leveraging knowledge from previous games.

(Parisotto, Ba, Salakhutdinov, ICLR 2016)
Deep Q-Network (DQN)

- DQN uses a deep function approximator to represent the state-action value function $Q(s,a)$.

- Q-function:
  \[
  Q^\pi(s, a) = \mathbb{E}\left[\sum_{t=0}^{H} \gamma^t r_t \mid s_0 = s, a_0 = a\right]
  \]
  expected future discounted reward when starting in a state $s$, executing $a$, and following policy $\pi$.

(Mnih et. al. 2014)
Deep Q-Network (DQN)

- The optimal Q-function (Bellman equation):
  \[ Q^*(s, a) = \mathbb{E}_{s' \sim T(\cdot|s,a)} \left[ r + \gamma \cdot \max_{a' \in \mathcal{A}} Q^*(s', a') \right] \]

- To train a DQN, the network’s loss is set to:
  \[ L^{(n+1)}(\theta^{(n+1)}) = \mathbb{E}_{s,a,r,s' \sim M(\cdot)} \left[ \left( r + \gamma \cdot \max_{a' \in \mathcal{A}} Q(s', a'; \theta^{(n)}) - Q(s, a; \theta^{(n+1)}) \right)^2 \right] \]

- where \( M(\cdot) \) is a uniform probability distribution over a replay memory -- a set of \((s, a, r, s')\) transition tuples seen during play.
Multitask DQN

• **Goal:** Given a set of source games $S_1, \ldots, S_N$, obtain a single multitask policy network that can play any source game.

• Use guidance from a set of expert DQN networks $E_1, \ldots, E_N$, where each $E_i$ is an expert specialized in source task $S_i$.

• Simply Q-learning a single DQN over many games at once does not work well:
  
  – The scale of Q-functions varies significantly between games, making learning unstable.

• **Alternative:** Attempt to match policies between expert networks and a single multitask network.
Policy Regression

- Transform each expert DQN into a policy network by a Boltzmann distribution:

\[ \pi_{E_i}(a|s) = \frac{e^{\tau^{-1}Q_{E_i}(s,a)}}{\sum_{a' \in \mathcal{A}_{E_i}} e^{\tau^{-1}Q_{E_i}(s,a')}} \]

- Define policy objective:

\[ \mathcal{L}_{policy}^i(\theta) = \sum_{a \in \mathcal{A}_{E_i}} \pi_{E_i}(a|s) \log \pi_{AMN}(a|s, \theta) \]

- Stable supervised training signal (the expert network output) to guide the multitask network.
Multitask as Model Compression

- Related to model compression, knowledge distillation (Ba et al. 2014, Hinton et al., 2015).

- A set of high complexity teacher networks guide a small network.

- Training data: we can sample either the expert network or the AMN action outputs.

- Empirically we observed that sampling from the AMN while it is learning gives the best results.

(Rusu et al. 2015)
Feature Regression

- Regress the features of the AMN towards the features of the expert network:

$$\mathcal{L}_{FR}^{i}(\theta, \theta_{f_i}) = \| f_i(h_{AMN}(s; \theta); \theta_{f_i}) - h_{E_i}(s) \|^2$$

hidden activations in the (pre-output) layer

- **Intuition**: Perfect regression implies that all the information in the expert features is contained in the multitask features.
Actor-Mimic Objective

• Combining both objectives, we obtain:

\[ \mathcal{L}_{ActorMimic}^i(\theta, \theta_{f_i}) = \mathcal{L}_{policy}^i(\theta) + \beta \mathcal{L}_{FR}^i(\theta, \theta_{f_i}) \]

• **Policy Regression**: A teacher (expert network) telling a student (AMN) how they should act (mimic expert’s actions).

• **Feature Regression**: A teacher telling a student why it should act that way (mimic expert’s “thinking” process).
Actor-Mimic Net in Action

• The multitask network can match expert performance on 8 games (we are extending this to more games).
Experimental Results

<table>
<thead>
<tr>
<th>Network</th>
<th>Atlantis</th>
<th>Boxing</th>
<th>Breakout</th>
<th>Crazy Climber</th>
</tr>
</thead>
<tbody>
<tr>
<td>DQN</td>
<td>Mean</td>
<td>57279</td>
<td>81.47</td>
<td>273.15</td>
</tr>
<tr>
<td></td>
<td>Max</td>
<td>541000</td>
<td>88.02</td>
<td>377.96</td>
</tr>
<tr>
<td>AMN</td>
<td>Mean</td>
<td>160204</td>
<td>75.746</td>
<td>346.85</td>
</tr>
<tr>
<td></td>
<td>Max</td>
<td>863050</td>
<td>81.620</td>
<td>374.58</td>
</tr>
<tr>
<td>100% × AMN/DQN</td>
<td>Mean</td>
<td>279.7%</td>
<td>92.98%</td>
<td>127.0%</td>
</tr>
<tr>
<td></td>
<td>Max</td>
<td>159.5%</td>
<td>92.73%</td>
<td>99.11%</td>
</tr>
</tbody>
</table>

• The multitask network can surpass expert performance in some games, such as Atlantis and Breakout, suggesting an inter-source-task transfer effect.

• The multitask network has the same network architecture as a single expert, yet can learn 8 games reasonably well.
Experimental Results

- The multitask network can surpass expert performance in some games, such as Atlantis and Breakout, suggesting an inter-source-task transfer effect.

- The multitask network has the same network architecture as a single expert, yet can learn 8 games reasonably well.

<table>
<thead>
<tr>
<th>Network</th>
<th>Atlantis Mean</th>
<th>Atlantis Max</th>
<th>Boxing</th>
<th>Breakout Mean</th>
<th>Breakout Max</th>
<th>Crazy Climber Mean</th>
<th>Crazy Climber Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>DQN</td>
<td>57279</td>
<td>541000</td>
<td>81.47</td>
<td>273.15</td>
<td>277.96</td>
<td>96189</td>
<td>117593</td>
</tr>
</tbody>
</table>
Transfer Learning

• Can the representations learnt on a set of source tasks generalize to new target games?

• We pre-train a network using Actor-Mimic on a set of 13 games and then use that as a weight initialization for a target task.
Learning as a Function of Time

Breakout: Performance after learning on

![Graph showing performance after learning on 500K frames vs 1 Million frames](image)
Learning as a Function of Time

Star Gunner: Performance after learning on

500K frames

1M frames
Quantitative Results

<table>
<thead>
<tr>
<th>Game</th>
<th>Breakout</th>
<th>1 mil</th>
<th>4 mil</th>
<th>7 mil</th>
<th>10 mil</th>
<th>Star Gunner</th>
<th>1 mil</th>
<th>4 mil</th>
<th>7 mil</th>
<th>10 mil</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DQN</strong></td>
<td>1.182</td>
<td>102.3</td>
<td>252.9</td>
<td>258.7</td>
<td></td>
<td>221.2</td>
<td>1084</td>
<td>3286</td>
<td>45322</td>
<td></td>
</tr>
<tr>
<td><strong>AMN-policy</strong></td>
<td>18.35</td>
<td>271.1</td>
<td>311.3</td>
<td></td>
<td></td>
<td>274.3</td>
<td>1667</td>
<td>31588</td>
<td>53642</td>
<td></td>
</tr>
<tr>
<td><strong>AMN-feature</strong></td>
<td>16.23</td>
<td>191.8</td>
<td>248.5</td>
<td>225.5</td>
<td></td>
<td>1405</td>
<td></td>
<td>50667</td>
<td>56839</td>
<td></td>
</tr>
<tr>
<td><strong>Kruell</strong></td>
<td>4302</td>
<td>7030</td>
<td>5949</td>
<td>6005</td>
<td></td>
<td></td>
<td>2323</td>
<td>5842</td>
<td>8468</td>
<td>11893</td>
</tr>
<tr>
<td><strong>DQN</strong></td>
<td>5827</td>
<td>6971</td>
<td>7854</td>
<td>7835</td>
<td></td>
<td>2583</td>
<td>143729</td>
<td>111074</td>
<td>123337</td>
<td></td>
</tr>
<tr>
<td><strong>AMN-policy</strong></td>
<td>5033</td>
<td>7582</td>
<td>8133</td>
<td>6923</td>
<td></td>
<td>1593</td>
<td>12421</td>
<td>15920</td>
<td>26379</td>
<td></td>
</tr>
<tr>
<td><strong>AMN-feature</strong></td>
<td>4.830</td>
<td>13.22</td>
<td>31.94</td>
<td>34.04</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Video Pinball</strong></td>
<td>3.502</td>
<td>9.215</td>
<td>18.66</td>
<td>23.02</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Robotank</strong></td>
<td>3.550</td>
<td>17.58</td>
<td>20.13</td>
<td>23.29</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- We pre-train a network using Actor-Mimic on 13 games.
- Speeds up learning in 3 out of the 7 target tasks tested.
- Causes negative transfer for one task.
- Provide small improvements for 4 games.
Talk Roadmap

• Hierarchical Deep RL

• Transfer Learning

• Learning with Memory
Reinforcement Learning with Memory

Learned External Memory

Read

Write

Action $a_t$

Reward $r_t$

Observation / State $O_t$

Differentiable Neural Computer, Graves et al., Nature, 2016;
Neural Turing Machine, Graves et al., 2014
Reinforcement Learning with Memory

Learned External Memory

Action

Write

$\alpha_t$

R

Learning 3-D game without memory

Chaplot, Lample, AAAI 2017

Differentiable Neural Computer, Graves et al., Nature, 2016;
Neural Turing Machine, Graves et al., 2014
Deep RL with Memory

Learned Structured Memory

Action $a_t$

Reward $r_t$

Observation / State $O_t$

Parisotto, Salakhutdinov, 2017
Random Maze with Indicator

- **Indicator**: Either blue or pink
  - If blue, find the green block
  - If pink, find the red block
- **Negative reward** if agent does not find correct block in N steps or goes to wrong block.

Parisotto, Salakhutdinov, 2017
Random Maze with Indicator

\[ M_t \]

Write

\[ a_t \]

Read with Attention

\[ w_t \]

\[ w_{t+1} \]

\[ M_{t+1} \]

\[ a_{t+1} \]

Parisotto, Salakhutdinov, 2017
Random Maze with Indicator
Building Intelligent Agents

- Learned External Memory
- Knowledge Base

Observation / State

Action $a_t$

Reason Communicate

Reward $r_t$

Write

Read
Building Intelligent Agents

Learning from Fewer Examples, Fewer Experiences